UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 09/748,098 | 12/21/2000 | Nicholas J. Kelsey | 20880-05093 | 4360 |

758 7590 05/03/2010
FENWICK & WEST LLP
SILICON VALLEY CENTER
801 CALIFORNIA STREET
MOUNTAIN VIEW, CA 94041

| EXAMINER | |
|---|---|
| HUISMAN, DAVID J | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2183 | |

| MAIL DATE | DELIVERY MODE |
|---|---|
| 05/03/2010 | PAPER |

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 09/748,098 | KELSEY ET AL. | |
| | Examiner | Art Unit | |
| | DAVID J. HUISMAN | 2183 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on 12 February 2010.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) 1-57 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-57 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 12 February 2010 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date 2/12/10.

4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

## DETAILED ACTION

1. Claims 1-57 have been examined.

### *Claim Objections*

2. Claim 1 is objected to because of the following informalities: In line 3 of the last paragraph, replace "between" with --from--. Appropriate correction is required.

### *Claim Rejections - 35 USC § 103*

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 1, 29-33, 42-43, and 45 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen et al., U.S. Patent No. 6,076,157 (herein referred to as Borkenhagen) in view of Gee et al., U.S. Patent No. 6,374,286 (herein referred to as Gee).

5. Referring to claim 1, Borkenhagen has taught a computer based system for switching between program contexts comprising:

a) a processor capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware. See Fig.4A and note that the processor includes thread switch (selection) hardware which selects between a first thread (thread 0) and a second thread (thread 1). Also, see column 7, lines 15-20, and note that the processor has an execution pipeline.

b) a first set of data storage devices capable of storing a first thread state of said processor. See Fig.4A, at least component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

c) a second set of data storage devices capable of storing a second thread state of said processor. See Fig.4A, at least component 444, and column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

d) a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching between execution of the first program thread directly to execution of the second program thread. See at least Fig.4A.

e) Borkenhagen has not taught that the thread-switching is caused at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. However, Gee has taught such a concept. See Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter "thread 29"). Each row in Fig.19 represents a chord, where the $N^{th}$ chord is "accessed" or "played" at the $N^{th}$ interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row

0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29 executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every $4^{th}$ chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every $4^{th}$ other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this

specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. And, it would have been further obvious to modify Borkenhagen to include the piano scheduling of Gee so that thread may be scheduled periodically. One would have been motivated to make such a modification to Borkenhagen in order to, at the very least, increase the scheduling flexibility of Borkenhagen. It should further be noted that even though Gee may be interpreted to not teach direct switching from a first thread to a second thread, the indirect switching is not a feature that would be imported into Borkenhagen. That is, Borkenhagen has already taught direct switching between two threads (i.e., there is no disclosure in Borkenhagen concerning some third thread being performed between the first and second threads). Therefore, given the teachings of Gee, one could directly switch between threads in a piano roll scheme. In other words, they important aspect of Gee is that threads can be scheduled at different frequencies. This teaching alone may be included in Borkenhagen.

f) Borkenhagen has further taught accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction includes in a program thread identified for execution by the predetermined fixed schedule. See Fig.4A and column 10, line 18, to column 11, line 17. Note that an inherently existing context number specifies which state register to access. For instance, if an instruction in thread 0 is executing, then the "0" specifies that state register 0 will be accessed.

6. Referring to claim 29, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said thread selection hardware in the pipelined processor switches between said first and second thread state after the end of the execution of a first program instruction in the first thread and before the beginning of the execution of a second program instruction. This is deemed inherent because thread A will execute for some amount of time and then a switch will occur to another thread. The switching marks the end of executing an instruction from thread A and the beginning of executing an instruction in thread B. And, clearly, the system must be in the second state before it can begin executing the second thread.

7. Referring to claim 30, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said processor is an embedded pipelined processor. See claim 14 of Borkenhagen. The processor, which is pipelined, is embedded in a system.

8. Referring to claim 31, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said first state is the state of the processor during the execution of the first program thread. See column 10, lines 18-56.

9. Referring to claim 32, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said second state is the state of the processor during the execution of the second program thread. See column 10, lines 18-56.

10. Referring to claim 33, Borkenhagen, as modified, has taught a system as described in claim 1. Gee has further taught that said processor switches between said first and second state by changing a state selection register. See column 20, lines 42-52, and column 32, lines 7-31. The hardware timer register (state selection register) is decremented each cycle until it gets to zero and then a thread switch occurs. Therefore, the processor switches between first and second state by changing a state selection register.

11. Referring to claim 42, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of restoring said second state of said processor during execution of said first program thread. See Fig.4A, component 450, and note that at some point during execution of the first thread, a thread switch will occur, and the second state is restored. That is, the switching to (restoring the state of) the second thread occurs during execution of the first thread.

12. Referring to claim 43, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of storing said second thread state of said processor during execution of said first program thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. This register may be set by any thread for itself or another thread.

13. Referring to claim 45, Borkenhagen, as modified, has taught a system as described in claim 1. Gee has further taught that the predetermined fixed schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. Gee has taught at least a fixed strict schedule where threads are switched based on the data in Fig.19.

14. Claims 2-4, 13, 16-17, 19-24, and 56-57 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy et al., U.S. Patent No. 6,542,991 (herein referred to as Joy), in view of McCrackin et al., "Using Horizontal Prefetching to Circumvent the Jump Problem", 1991 (herein referred to as McCrackin).

15. Referring to claim 17, Joy has taught a computer based system for switching between program contexts comprising:

a) a pipelined processor (Fig.3, component 300; column 8, lines 14-67) capable of having a first program thread and a second program thread in an execution pipeline coupled to thread selection hardware (see Fig.6 and note the "thread select logic"; column 13 lines 5-23, column 15 lines 4-7), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage in the set of stages (see claim 1, for instance, and note the existence of a pipeline). It should be noted that pipelines inherently include stages which perform different tasks, and in which a different instruction may be executing at one time.

b) a first set of data storage devices capable of storing a first thread state of said pipelined processor. See Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0). Also, instruction cache 330 stores instructions for thread 0.

c) a second set of data storage devices capable of storing a second thread state of said pipelined processor. See Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1). Also, instruction cache 330 stores instructions for thread 1. Note that thread 0 and thread instructions are each stored in a unique location in memory (clearly two sets of data cannot be stored in the same physical location).

d) wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to a hardware thread scheduler identifying the second thread state for execution by said pipelined processor. See Fig.6, column 15, lines 4-7, and the example shown in column 3, lines 33-51, and column 17, lines 1-6. Also, see Fig.4A and column 10, line 18, to column 11, line 17. Note that an inherently existing context number specifies which state register to access. For instance, if an instruction in thread 0 is executing, then the "0" specifies that state register 0 will be accessed.

e) Joy has not taught that the processor switches from said first thread state directly to said second thread state between consecutive instruction cycles <u>without incurring a time penalty</u>. However, McCrackin has taught such a concept. See page 1287, column 2, section II, paragraphs 2-3. Specifically, resources of the execution and fetch units are duplicated for as many thread contexts that exist. Such duplication allows for elimination of context switching overhead because instead of having to spend time saving and restoring state information in shared resources each time a context is switched (as is the case in Joy, column 15, lines 1-7), one

of the duplicated (non-shared) contexts is merely selected to switch threads, i.e., no saving/restoring in shared resources is required. As a result, in order to eliminate time penalties associated with thread switching, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that resources are duplicated for each thread instead of sharing resources and having to perform time-costly saves and restores each time a thread switch occurs.

f) Joy, as modified, has further taught the hardware thread scheduler is configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves data from the first set of data storage devices or the second set of data storage devices. See column 3, lines 33-51, and note that Joy's thread-switching may be of the oblivious type, in which threads are switched every N cycles without notification of stalling. Note that depending on the thread schedule, the appropriate data storage device is selected to retrieve instructions from. All schedules are predetermined as they are determined prior to executing the application.

16. Referring to claim 2, Joy, as modified, has taught a system as described in claim 17, wherein said first thread state is the thread state of the processor during the execution of the first program thread. See Joy, Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

17. Referring to claims 3, Joy, as modified, has taught a system as described in claim 17, wherein said second thread state is the thread state of the processor during the execution of the

second program thread. See Joy, Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

18. Referring to claim 4, Joy, as modified, has taught a system as described in claim 17, wherein said processor switches between said first and second thread state by changing a state selection register. See Joy, Fig.5 and column 13, lines 5-64. The thread select logic includes a flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread.

19. Referring to claim 13, Joy, as modified, has taught a system as described in claim 17, wherein said processor is capable of restoring said second thread state of said processor during execution of said first program thread. See Joy, column 6, lines 15-35. Clearly, a thread is executed until a switch signal is given. Therefore, switching to (restoring a) thread occurs during execution of another thread.

20. Referring to claim 16, Joy, as modified, has taught a system as described in claim 17, wherein the execution schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. From column 3, lines 28-51, switching every N cycles is considered a fixed strict schedule, i.e., a thread switch must occur every N cycles.

21. Referring to claim 19, Joy has taught a computer based method for switching between program contexts in a multithreading pipelined processor (Fig.3; column 8, lines 14-67) having a hardware thread selector (Fig.6) and an execution pipeline (see claim 1 of Joy, for instance), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage of the set of stages (it should be noted that pipelines

inherently include stages which perform different tasks, and in which a different instruction may be executing at one time), the method comprising:

a) storing a first context of said pipelined processor in a first set of data storage devices, the first context corresponding to a first program thread. See Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

b) storing a second context of said pipelined processor in a second set of data storage devices, the second context corresponding to a second program thread. See Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

c) responsive to a predetermined fixed schedule allocating processing time to the first program thread and to the second program thread, switching the pipelined processor from executing the first program thread to executing the second program thread. See Fig.6, column 15, lines 4-7, and the example shown in column 3, lines 33-51, and column 17, lines 1-6. Note that all scheduling algorithms are predetermined fixed schedules. That is, the scheduling algorithms were inherently implemented prior to execution of the program (i.e., predetermined); otherwise threads could not be scheduled according to those algorithms. Also, they are fixed. The oblivious algorithm is always the oblivious algorithm, for instance.

d) Joy has not taught switching the pipelined processor from executing the first program thread to executing the second program thread <u>between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty</u>. However, McCrackin has taught such a concept. See page 1287, column 2, section II, paragraphs 2-3.

Specifically, resources of the execution and fetch units are duplicated for as many thread contexts that exist. Such duplication allows for elimination of context switching overhead because instead of having to spend time saving and restoring state information in shared resources each time a context is switched (as is the case in Joy, column 15, lines 1-7), one of the duplicated (non-shared) contexts is merely selected to switch threads, i.e., no saving/restoring in shared resources is required. As a result, in order to eliminate time penalties associated with thread switching, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that resources are duplicated for each thread instead of sharing resources and having to perform time-costly saves and restores each time a thread switch occurs.

d) Joy, as modified, has further taught switching the pipelined processor from executing the first program thread to executing the second program thread by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the first set of data storage devices or the second set of data storage devices. Recall from above that Joy has taught thread switching. An example is shown in column 3, lines 33-51, and column 17, lines 1-6). Context numbers (e.g., thread IDs) inherently exist so that the processor can track which thread is being switched out (and so it is not selected again immediately) and which threads are available for being switched in. The context number of the thread selected dictates which of the replicated devices data is retrieved from.

22. Referring to claim 20, Joy, as modified, has taught a system as described in claim 19, wherein the switching comprises changing a state selection register included in the hardware thread selector. See Joy, Fig.5 and column 13, lines 5-64. The thread selector logic includes a

flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread. When a thread is switched, then a new bit in the register would be set while the previously set bit is reset.

23. Referring to claim 21, Joy, as modified, has taught a method as described in claim 19, further comprising identifying which of the said program threads said processor executes according to an execution schedule. See Joy, column 3, lines 33-51.

24. Referring to claim 22, Joy, as modified, has taught a method as described in claim 21, further comprising allocating available processing time of the processor among at least the first and second threads according to the execution schedule. See Joy, column 3, lines 33-51.

25. Referring to claim 23, Joy, as modified, has taught a method as described in claim 22, wherein the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. See Joy, column 3, lines 33-51, and note that each thread will execute for N cycles in one scheduling embodiment.

26. Referring to claim 24, Joy, as modified, has taught a method as described in claim 23, wherein at least one quanta corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. See Joy, column 3, lines 33-36.

27. Referring to claim 56, Joy, as modified, has taught the method of claim 19, wherein switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty comprises: identifying a context number associated with an instruction included in the second program thread, the context number

identifying the second set of data storage devices; communicating the context number associated with the second program thread to the execution pipeline; and loading instructions from the second set of data storage devices into the execution pipeline. The examiner asserts that each of these steps is inherent in the combined Joy and McCrackin system. If the system switches contexts, then a signal selecting a second context is inherently issued such that the second thread's instructions may be retrieved and loaded into the pipeline. If N contexts exist, as is the case in McCrackin, then one of N threads must be specified.

28. Referring to claim 57, Joy, as modified, has taught the system of claim 17, wherein said thread selection hardware further communicates the context number associated with the second program thread to the execution pipeline and loads instructions from the second set of data storage devices into the execution pipeline. The examiner asserts that each of these steps is inherent in the combined Joy and McCrackin system. That is, each instruction is to access only one thread's context data or state. Hence, in order to select the appropriate state, the context number must be used to select one of the states. Also, when the second thread is active, instructions are loaded from the data storage devices into the pipeline for execution. Note that the data storage devices include instruction memories where the instructions are stored.

29. Claims 5-10, 18, and 25-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Ramakrishnan et al., U.S Patent No. 6,085,215 (herein referred to as Ramakrishnan).

30. Referring to claim 5, Joy, as modified, has taught a system as described in claim 17. Joy has not taught that said hardware thread scheduler includes a thread identifier for identifying at

least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time. In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real-time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

31. Referring to claim 6, Joy, as modified, has taught a system as described in claim 5. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

32. Referring to claim 7, Joy, as modified, has taught a system as described in claim 5. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time

(NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

33. Referring to claim 8, Joy, as modified, has taught a system as described in claim 5. Ramakrishnan has further taught that said thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

34. Referring to claim 9, Joy, as modified, has taught a system as described in claim 8. Ramakrishnan has further taught that said thread scheduler regularly schedules NRT threads to be executed. See the abstract and note that there may be a plurality of NRTs for scheduling. They are scheduled for minimum times throughout the entire execution process.

35. Referring to claim 10, Joy, as modified, has taught a system as described in claim 5. Joy has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.3, component 330. The instruction cache (I$) will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See column 9, line 66. The main memory will be fetched from during the time that the instructions needed are not in the cache.
c) wherein said first fetch period is substantially shorter than said second fetch period. Fetching from a cache is shorter than fetching from main memory, as is known in the art.

36. Referring to claim 18, Joy, as modified, has taught a system as described in claim 5. Joy has further taught that said time quanta is exactly one instruction cycle. See column 17, lines 1-6, and note that Joy has taught a time quanta of N cycles, where N includes 1.

37. Referring to claim 25, Joy, as modified, has taught a method as described in claim 21. Joy has not taught identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. However, Ramakrishnan has taught the concept of real-time threads and general (non-real-time). See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads. This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system gets important work done while preventing starvation of threads (since all threads get some time to process) and offering a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to identify at least one HRT and at least one NRT thread, as taught by Ramakrishnan.

38. Referring to claim 26, Joy, as modified, has taught a method as described in claim 25. Ramakrishnan has further taught scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

39. Referring to claim 27, Joy, as modified, has taught a method as described in claim 25. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

40. Referring to claim 28, Joy, as modified, has taught a method as described in claim 25. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

41. Claims 11-12 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and Ramakrishnan, and further in view of the examiner's taking of Official Notice.

42. Referring to claim 11, Joy, as modified, has taught a system as described in claim 10. Joy has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's instruction cache such that it is implemented in SRAM.

43. Referring to claim 12, Joy, as modified, has taught a system as described in claim 10. Joy has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore <u>flash main memory</u> based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's main memory such that it is a flash-based main memory.

44. Claim 14 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Borkenhagen.

45. Referring to claim 14, Joy, as modified, has taught a system as described in claim 17. Joy has not explicitly taught that said processor is capable of storing said second thread state of said processor during execution of said first program thread. However, Borkenhagen has taught that a thread switch control register may be implemented for each thread for holding a state of that thread and that the control state for the second thread may be stored during execution of the first thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include the control register of Borkenhagen as part of the thread state, where the thread state of the second thread is stored during execution of the first thread.

46. Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Levy et al., U.S. Patent No. 6,314,511 (herein referred to as Levy).

47. Referring to claim 15, Joy, as modified, has taught a system as described in claim 17. While Joy has taught that said first set of data storage devices comprises registers (see column 8, lines 27-44, and column 3, lines 3-10), Joy has not taught that the registers are shared by a plurality of threads. Instead, Joy has taught that each thread gets its own register file. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs, as taught by Joy. In addition, by

sharing registers, the total size of the register file is reduced (since you don't have to have a separate register file for each thread) without sacrificing performance. See Levy, column 8, line 65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the registers are shared by the threads instead of replicated.

48. Claims 34-39 and 48-55 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and further in view of Ramakrishnan.

49. Referring to claim 34, Borkenhagen, as modified, has taught a system as described in claim 1. Borkenhagen has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time (i.e., in a preselected maximum time). In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See

column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

50. Referring to claim 35, Borkenhagen, as modified, has taught a system as described in claim 34. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

51. Referring to claim 36, Borkenhagen, as modified, has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

52. Referring to claim 37, Borkenhagen, as modified, has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

53. Referring to claim 38, Borkenhagen, as modified, has taught a system as described in claim 37. Ramakrishnan has further taught that said hardware thread scheduler regularly schedules NRT threads to be executed. See the abstract and note that there may be a plurality of NRTs for scheduling.

54. Referring to claim 39, Borkenhagen, as modified, has taught a system as described in claim 34. Borkenhagen has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.1, component 150. The cache will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See Fig.1, component 140. The main memory will be fetched from during the time that the instructions needed are not in the cache.

c) wherein said first fetch period is substantially shorter than said second fetch period. See column 3, lines 13-17. Fetching from a cache is shorter than fetching from main memory.

55. Referring to claim 48, Borkenhagen, as modified, has taught a method as described in claim 46. Borkenhagen has not taught identifying which of the said program threads said processor executes according to a hard real-time (HRT) execution schedule. However, Ramakrishnan has taught the concept of real-time threads. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system. In such a system, real-time threads are given priority over general threads (note that Borkenhagen has also taught using

priority with threads in the abstract). This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks, while allowing important work to get done. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen to be used in a time-critical environment and to identify which of the program threads the processor executes according to an HRT execution schedule, as taught by Ramakrishnan.

56. Referring to claim 49, Borkenhagen, as modified, has taught a method as described in claim 48. Gee has further taught allocating available processing time of the processor among at least the first and second threads according to the predetermined fixed execution schedule. Please recall the rejection of claim 46 and note that time is allocated to periodic threads according to the schedule fixed by the structure of Fig.19.

57. Referring to claim 50, Borkenhagen, as modified, has taught a method as described in claim 49. Gee has further taught that the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. Again, see the rejection of claim 46. Each thread is allocated a number of cycles (quanta) in which to execute. This number is associated with the timeout value of the hardware timer.

58. Referring to claim 51, Borkenhagen, as modified, has taught a method as described in claim 50. Gee has further taught that at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. Please recall the rejection of claim 46 and Fig.19 of Gee.

59. Referring to claim 52, Borkenhagen, as modified, has taught a method as described in claim 48. Ramakrishnan has further taught that identifying further comprises identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads, which are the less time-critical threads.

60. Referring to claim 53, Borkenhagen, as modified, has taught a method as described in claim 52. Ramakrishnan has further taught scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

61. Referring to claim 54, Borkenhagen, as modified, has taught a method as described in claim 52. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

62. Referring to claim 55, Borkenhagen, as modified, has taught a method as described in claim 52. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

63. Claims 40-41 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and Ramakrishnan, and further in view of the examiner's taking of Official Notice.

64. Referring to claim 40, Borkenhagen, as modified, has taught a system as described in claim 39. Borkenhagen has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's instruction cache such that it is implemented in SRAM.

65. Referring to claim 41, Borkenhagen, as modified, has taught a system as described in claim 39. Borkenhagen has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy

based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's main memory such that it is a flash-based main memory.

66. Claim 44 is rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and further in view of Levy.

67. Referring to claim 44, Borkenhagen, as modified, has taught a system as described in claim 1. While Borkenhagen has hinted at sharing of resources (column 5, lines 56-57), Borkenhagen has not explicitly taught that said first set of storage devices comprises registers shared by a plurality of threads. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs. In addition, by sharing registers, the total size of the register file is reduced (since you don't have to have a separate register file for each thread) without sacrificing performance. See Levy, column 8, line 65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen such that the registers are shared by the threads.

68. Claims 46-47 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and further in view of McCrackin.

69. Referring to claim 46, Borkenhagen has taught a computer based method for switching between program contexts in a multithreading pipelined processor (see Fig.4A and the abstract) having a hardware thread selector (see Fig.4A) and an execution pipeline (see column 7, lines 15-20 and note that the processor has an execution pipeline), the method comprising:

a) storing a first context of said processor in a first set of data storage devices comprising a first thread state corresponding to a first program thread. See Fig.4A, at least component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

b) storing a second context of said processor in a second set of data storage devices comprising a second thread state corresponding to a second program thread. See Fig.4A, at least component 444, and column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

c) switching the processor directly from the first thread state to the second thread state by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector responsive to a context number associated with an instruction included in the first thread state or in the second thread state identifying the first set of data storage devices of the second set of data storage devices. See at least Fig.4A and column 10, line 18, to column 11, line 17. Note that an inherently existing context number specifies which state register to access. For instance, if an instruction in thread 0 is executing, then the "0" specifies that state register 0 will be accessed.

d) Borkenhagen has not taught that the decoupling and coupling occur at a fixed time according to a predetermined fixed execution schedule, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to the second number of cycles. However, Gee has taught such a concept. See Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter "thread 29"). Each row in Fig.19 represents a chord, where the $N^{th}$ chord is "accessed" or "played" at the $N^{th}$ interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row 0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29 executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's

intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every 4th chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every 4th other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is

not equal to said second number of cycles. And, it would have been further obvious to modify Borkenhagen to include the piano scheduling of Gee so that thread may be scheduled periodically. One would have been motivated to make such a modification to Borkenhagen in order to, at the very least, increase the scheduling flexibility of Borkenhagen. It should further be noted that even though Gee may be interpreted to not teach direct switching from a first thread to a second thread, the indirect switching is not a feature that would be imported into Borkenhagen. That is, Borkenhagen has already taught direct switching between two threads (i.e., there is no disclosure in Borkenhagen concerning some third thread being performed between the first and second threads). Therefore, given the teachings of Gee, one could directly switch between threads in a piano roll scheme. In other words, they important aspect of Gee is that threads can be scheduled at different frequencies. This teaching alone may be included in Borkenhagen.

e) Borkenhagen, as modified, has not taught switching between threads without incurring a time penalty. However, McCrackin has taught such a concept. See page 1287, column 2, section II, paragraphs 2-3. Specifically, resources of the execution and fetch units are duplicated for as many thread contexts that exist. Such duplication allows for elimination of context switching overhead because instead of having to spend time saving and restoring state information in shared resources each time a context is switched, one of the duplicated (non-shared) contexts is merely selected to switch threads, i.e., no saving/restoring in shared resources is required. As a result, in order to eliminate time penalties associated with thread switching, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen

such that resources are duplicated for each thread instead of sharing resources and having to perform time-costly saves and restores each time a thread switch occurs.

70. Referring to claim 47, Borkenhagen, as modified, has taught a method as described in claim 46. Gee has further taught that the switching comprises changing a state selection register included in the hardware thread selector. See column 20, lines 42-52, and column 32, lines 7-31. The hardware timer register (state selection register) is decremented each cycle until it gets to zero and then a thread switch occurs. Therefore, the processor switches between first and second state by changing a state selection register.

***Response to Arguments***

71. Regarding the amendment made to claim 1 and the argument therefor, the examiner asserts that applicant has merely added an inherent feature to the claim. In Borkenhagen, a state register exists for each thread. Hence, when an instruction from thread 0 is executing, state register 0 must be selected. When an instruction from thread 1 is executing, state register 1 must be selected. Hence, the system must inherently track the currently executing thread by latching the context number in some fashion. Otherwise, there would be no way to determine which register is to be selected.

72. Also, regarding Gee, even if the frequency scheduling is based on priority, this does not mean Gee has not taught the predetermined fixed schedule of applicant's claim 1. Borkenhagen supports thread priorities, which are stored in the state registers. See columns 10-11. Therefore, two or three threads may exist, each having a priority, and they may be scheduled according to

the piano roll scheduler, which dictates that a first thread gets processing time every N cycles, and a second thread gets processing time every M cycles.

73. Regarding the amendment made to claim 46 and the argument therefor, the examiner asserts that the argument is not persuasive for the same reason the argument for claim 1 is non-persuasive. Also, not that an additional reference has been added to the claim 46 rejection to address the "time penalty" limitation.

74. Regarding the amendment made to claim 17 and the argument therefor, the examiner disagrees that Joy has not taught a “predetermined execution schedule” as claimed. Clearly, Joy has taught such a thing as the schedules of Joy are implemented prior to execution of the threads; otherwise the threads could not be scheduled according to that schedule. For instance, the oblivious schedule set forth in column 3, lines 33-38, is always an oblivious schedule. In that sense, it is predetermined. The examiner would agree that Joy has not taught the specifics of the schedule set forth in the claim, but claim 17 does not recite those specifics.

75. The examiner agrees that Joy has not taught “without incurring a time penalty” because Joy must save/restore register contents. However, this is the reason for combining Joy with McCrackin, which teaches replicating register resources so that the copying/saving, and consequently, the associated time penalty, can be avoided. It should be noted that the context number limitation added to claim 17 is again inherent. Any system switching between N threads and selecting an associated one of N resources must track the context number so it can determine

which resource to select for use by the current thread. Without the context number, there would be no way to determine which resource(s) to access.

76. The remaining arguments/rejections are not persuasive for the same reasons the addressed arguments above were not persuasive. Please see the rejections for additional details.

***Conclusion***

77. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183